Typescript Fundamentals Beginners Tutorial

`TypeScript` is an open-source object-oriented language developed and maintained by `Microsoft`. It is a typed superset of JavaScript that compiles to plain JavaScript. Will learn/review the `new keywords-let, const` and `function/object syntax` also shows how the new `class, interface, module syntax` can be used with JavaScript library like Angular, React, Node. TypeScript is pure object oriented with classes, interfaces, modules, import, export and statically typed like C# or Java. The popular JavaScript framework Angular 2.x and above versions are written in TypeScript. Mastering TypeScript can help developers and programmers to write object-oriented programs and have them compiled to JavaScript, both on the server side and client side.

Prerequisites for current course / What you need to know

`TypeScript` introduces us to the newest language features that can be used in JavaScript code. Due to this, I assume you have some working knowledge/understanding of JavaScript. If you haven't, will recommend watching JavaScript Essentials Tutorial for beginners. Basic knowledge of JavaScript is required, having a good understanding of OOP concepts and advanced JavaScript is an added advantage. Finally, you will be using a bit of Angular.js, React.js, and Node.js but nothing that requires deep/advanced knowledge.

Topics include

Section 1. Course Introduction

1.1. Welcome

Hi All, I'm **`Dinanath Jayaswal, Senior UI/Web Developer and Adobe Certified Expert Professional`**, I wanna welcome you to `Typescript fundamentals beginners tutorial`. In this course/tutorial will go over how to start with using the Typescript features right with the bang. Will dive into the specifics of Typescript and new features it includes. We'll look at new keywords, syntax, and operators that can be used to simplify code structure. At the end we'll take a look at class, module syntax in Typescript and how that can be used immediately with very popular JavaScript library/frameworks like Angular, React, Node, etc. Let us explore and learn some pretty exciting stuff so let's go ahead and get started with `Typescript`.

`Typescript` will introduce us to the newest language features that can be used in JavaScript code - The Modern JavaScript.

1.2. Who is this for?

This course is for anyone and everyone, Almost everyone! who is interested in boost skills and further career - by learning new latest programming/coding standards/features/syntaxes/keywords introduced in the latest version of JavaScript to become a hi-tech developer. Programmers/Developers coming from any language wants to make a career in Web Technologies can learn Typescript.

Section 2. Introduction to Typescript

2.1. What is Typescript?

- Typescript is Open source/free, exciting, dynamic new development language, library from `Microsoft`
- TypeScript is a strongly typed, object oriented, compiled language
- It was designed and developed by `Anders Hejlsberg (designer of C#) at Microsoft`
- `Typed superset of JavaScript` which Compiles to plain JavaScript
- It consists of variables Data Types string, number, boolean, etc. but it is optional
- One can write safer/meaningful/easily maintainable/scalable code
- The developer can Identify mistakes during development/compilation step (scope of the variable, function parameter, variable datatype mismatch, etc.)
- Angular latest version uses typescript as an official/primary development language
- `Typescript = ECMAScript 5 + 6 + 7 + Latest features`
- TypeScript is JavaScript plus some additional features, Typescript is a language that is meant to be compiled into JavaScript:
    - `Typescript => JavaScript` OR `.ts => .js`

2.2. Why Typescript?

- Typescript got a similar syntax to JavaScript
- Typescript integrates easily into JavaScript projects
- TypeScript is extended JavaScript and superior to its other counterparts like `CoffeeScript` and `Dart` programming languages in a way that extends JavaScript
- Currently, it is used in many large projects and many libraries written/developed fully in Typescript:
    - `Google Angular 2.x` and greater version,
    - `WinJS` (The Windows library for JavaScript)
- The TypeScript transpiler provides the error-checking feature
- TypeScript comes with an optional static typing and types inference system through the `TLS (TypeScript Language Service)`
- Prevent bugs (data type)
- TypeScript supports Object Oriented Programming concepts like classes, interfaces, inheritance, etc.
- Increases the code quality, readability and makes it an easy to maintain and refactor the code base
- Also integrated with task runners like Grunt & Gulp
- Rapid growth and use

2.3. Components of TypeScript

1. `Language` – syntax, keywords, and type annotations
2. `TSC TypeScript Compiler` – converts TypeScript to JavaScript equivalent
3. `TLS TypeScript Language Service` – Supports editor operations like static typing and type inference system, statement completion, code formatting, etc.

Section 3. Setting up Typescript Development Environment

3.1. Installing Node, NPM and Typescript

- Install node from https://nodejs.org/en/
- Open up your terminal window/command prompt (`cmd, git bash, node command prompt`) to run some of node commands
- Verify and check node installation/version by using command: `node -v`
- install typescript globally by using command: `npm install -g typescript`
- Verify typescript Installation by command: `tsc` (tsc = typescript compiler)
- Verify typescript Installation and version both with command: `tsc -v` or `tsc --version`
- Install and use some text editor or IDE (Integrated Development Environment) like `Notepad++, Sublime Text, Adobe Brackets, Visual Studio Code` to type code

3.2. Write first Typescript code/program

- Create a new folder to store all typescript/.ts example files
- Create a first .ts file and type basic javascript/typescript or valid javascript code
- Now by using node/npm command compile .ts file into .js, command: `tsc file.ts`
- Use type annotation to eliminate any data variable type error: `let name:string = 'Dinanath';`
- We can use/run .js files in browser or in NodeJs (node command to run any .js file: `node file.js`), NodeJs can not run/understand .ts file
- Use automatic compilation method using watch mode: `tsc file.ts --watch`

> **Note**: Anything that is JavaScript code/programm is legal TypeScript

> **Syntax & Example**: `Typescript`

```
// by default typescript consider all files in a folder as a module, so variables
defined in one file will be checked on the fly and throws an error: Cannot
redeclare block-scoped variable - to solve issue use export {}

export {};

// write valid javascript, basic javascript/typescript
console.log('Welcome to typescript');
// ------------------------------

//type annotation to eliminate any data variable type error
let name:string = 'Dinanath';
//name=123; // error
console.log('My Name is: ' + name);
```

3.3. TypeScript Playground

Write and test Typescript without download or install anything: https://www.typescriptlang.org/play

3.4. ScratchJS

- Alternatively, Simply we can add/install a `Google Chrome Extention` named `Scratch JS`, an add-on for DevTools which integrates both the Traceur and Babel transpilers, allowing us to test out the new JS features coming with ES6/ES2015
- `Scratch JS` also supports `CoffeeScript`, `LiveScript` and more compile-to-JS languages will be added soon
- To install `Scratch JS` as a `Google Chrome Extention`:
    - In google chrome web store https://chrome.google.com/webstore/category/extensions?hl=en-GB or at https://www.google.com/ search for Scratch JS
    - From searched result select Scratch JS and click -> ADD to Chrome
- Once extension installed properly, open developer tool (F12, Fn12, Right Click on web page Inspect), click on ScratchJS - at left side type ES6 code and right side check ES5 output

Section 4. Typescript latest features-concepts

4.1. TypeScript - Basic Syntax

Syntax is nothing but a set of rules for writing programs. A TypeScript program simply composed of:

- Statements and Expressions
- Variables
- Functions
- Comments
- Modules

4.2. Variable Declarations

4.2.1. Variable

- A variable is `container to store/hold the data/information`
- A variable is a kind of data holder where we can store some value for programming or calculation purpose
- A JavaScript variable is simply a `name of the storage location (named containers/named storage)` for data
- Variables are symbolic names for values
- Variables are used to store data of different types like a string of text, numbers, boolean values like true/false, an array of data, etc.
- The data or value stored in the variables can be set, updated, and retrieved whenever needed
- TypeScript follows the same variable declarations rules as JavaScript, Variables can be declared using: `var`, `let`, and `const`

4.2.2. `var` Variable Declarations

- Variables are declared traditionally using the keyword: `var` keyword
- Many things what we do in plain JavaScript works exactly same with typescript (like declaring a variable with `var`)
- Vanilla JavaScript `var` and `scope` are tricky, they hoist also it has `global` and `functional` scope but not `block` level scope
- var keyword variables `can be re-declared` n number of times
- The `assignment operator (=)` is used to assign value to a variable, like this: `var varName = value;` or `var firstName = 'JavaScript';`
- **Example**: Variables are like `box or an envelope` which we use to `organize various kinds of stuff` and put a `label` on each box or an envelope
- **Example**: Variable declaration and assignment is just like `Maths & Algebra: x = 10;` and in JavaScript we write `var x = 10;`

> **Syntax & Example**: `Typescript`

```
export {};

var name: string = 'Dinanath';
var age: number = 35;
var isDeveloper: boolean = true;
console.log('My details are : ' + name + ' ' + age + ' ' + isDeveloper);

var name = 'Dino';
console.log('My name is : ' + name);
```

4.2.3. `let` Variable Declarations

- Typescript encourages programmers to use `let` and `const` keyword-based variables
- `let` and `const` variables have `block` level scope also they `can not be re-declared multiple times`
- To `deal with scope` in JavaScript we have new keyword named `let` used to declare variables
- We use the let keyword to `create block scoping` in JavaScript in locations where we weren't able to do so before
- `Let keyword block scope` - let keyword variables exist only inside `block { }`scope
- let keyword variables cannot be used before the declaration (`never hoisted`)
- let keyword variables `can not be re-declared`

> **Syntax & Example**: `Typescript`

```
export {};

let name: string = 'Dinanath';
let age: number = 35;
let isDeveloper: boolean = true;
console.log('My details are : ' + name + ' ' + age + ' ' + isDeveloper);
```

```
// let variables not be re-declared
// let name: string = 'Dino';
```

4.2.4. `const` Variable Declarations (also known as `"immutable variables"`)

- Typescript encourages programmers to use `let` and `const` keyword-based variables
- `let` and `const` variables have `block` level scope also they `can not be re-declared multiple times`
- Like the let keyword, we can also use `const` an alternative for declaring variables
- const is short for `constant` allows us to set constant variables which `shouldn't be reassigned`
- We can not change the value of const variable, so use const to define `fixed set of values` also to `protect the values` of certain variables
- Creating/Using an existing const variable again or re-assigning value to const variable generates `TypeError, SyntaxError`
- the const keyword is used to define/create `variable with read-only` constants
- const must have some value `while declaration/initialized`
- const value is fixed, `not to change/re-assigned` in future
- const is also having `block scope`
- the const value `never hoisted`
- However, we can still change object properties or array elements

> **Syntax & Example**: `Typescript`

```
export {};

//const name; //error - const must have some value `while declaration/initialized`
// const name;

const name: string = 'Dinanath';
console.log('My name is : ' + name );

//error - const value is fixed, `not to change/re-assigned` in future
// const name: string = 'Dino';
```

4.2.5. let vs const

- If variables have `one time assignments (fixed value, not changed in future)` use const
- If value re-assignments/`value updation` requires use `let`
- Const variables need value at the time of definition; `const name='Dinanath';`
- Let variables can be defined/initialized with empty value; `let name;`
- Const variables are `immutable, not changeable` variables
- Let variables are `mutable, can be changed or re-assigned`

4.3. Type Annotations / Type System

- JavaScript is not a typed language, However, TypeScript is a typed language, where we can specify the type of the variables, function parameters and object properties like a number, string, boolean, etc.
- We can specify the type using `: Type` after the name of the variable, parameter or property, It is advisable to use `annotation method to specify data type while declaring the variable` (Example: let firstName: string = 'Dinanath')
- Typescript does not allow to assign a different type of data to a variable
- `Type annotation` method helps to eliminate any errors related to variable data type mismatch
- The `any` data type is the supertype of all types in TypeScript which denotes a `dynamic type`

Advantage/Use of Type Annotations / Type System:

1. **Static Type checking errors** (Data type mismatch error)
2. **Getting accurate intellisense help for specific data type** (Example: specific data type variable will show only specific/related methods and properties etc.)

> **Syntax & Example**: `Typescript`

```
export {};

let isCompleted: boolean = false;
let id: number = 101;
let firstName: string = 'Dinanath';
let lastName: string = 'Jayaswal';

console.log(`User Details: ${firstName} ${lastName} ${isCompleted} ${id} `)

let fullName;
function showFullName(_fn: string ,_ln: string ) {
  let fullName = _fn + ' ' + _ln;
  console.log(`My FullName: ${fullName}`);
}

showFullName('Dino','Jas');
```

4.4. Data type `any`

- Typescript introduced a new data type named `any`
- A variable with `any` type can hold value with any data type (like the first variable holds string than assign number or boolean type)
- For data type `any` there is no compile-time checking
- The `any` data type is the supertype of all types in TypeScript which denotes a `dynamic type`

> **Syntax & Example**: `Typescript`

```
let data1:string = 'Data1'

//error - string type can not accept number or other data
// data1 = 50;
```

```
// any data type variable can store any type of value, there are no data checking
let data2:any = 'Data2';
data2 = 50;
data2 = true;
data2 = ['Hi','Hey','Good','Hello'];

console.log(data2);
```

4.5. Data type `array`

- Arrays in Typescript works in the same way as they work in JavaScript
- Same as JavaScript, TypeScript allows us to deal with arrays of values
- There are two ways to write/declare an array:
    1. let cities: string[];
        - let arrFrameworks: string [] = ['JavaScript','jQuery','Angular','React'];
    2. let cities: Array<string>
        - let arrLanguages: Array<string> = ['C','C++','Java','Ruby','Phthon'];
        - In the above example, we declared an array of strings by assigning it the string type (Now TypeScript will make sure the array contains only strings)

> **Syntax & Example**: `Typescript`

```
export {};

let arrFrameworks: string [] = ['JavaScript','jQuery','Angular','React'];
console.log('arrFrameworks:',arrFrameworks);


let arrLanguages: Array<string> = ['C','C++','Java','Ruby','Phthon'];
console.log('arrLanguages',arrLanguages);
```

4.6. Tuples

- Typescript provides a new special type of array called `Tuples which can store multiple data types based values`
- Tuples represent a heterogeneous collection of values `(multiple types of values in one collection/collection of values of various types)`
- Tuples enable storing multiple fields of different types
- let personDetail: [string, number] = ['Dinanath', 35];

> **Syntax & Example**: `Typescript`

```
export {};

let arrFrameworksDetails:  [string,number] = ['JavaScript',7.0];
```

```
console.log('arrFrameworksDetails:',arrFrameworksDetails);

console.log('arrFramework Name:',arrFrameworksDetails[0]);
console.log('arrFramework version:',arrFrameworksDetails[1]);
```

4.7. Enumeration

- Enums are used to define a set of named constants which can be used to document intent or to create a set of different cases (collection of related values that can be numeric or string values)
- Enums defined or start with `enum variableName` keyword
- There are three types of enums:
    1. Numeric enum (number-based enums i.e. they store string values as numbers)
    2. String enum (initialized with string values)
    3. Heterogeneous enum (contain both string and numeric values)

> **Syntax & Example**: `Typescript`

```
export {};

enum RGBColor { Red, Green, Blue };
console.log('RGBColor:', RGBColor);
console.log('RGBColor.Green:', RGBColor.Green);

let colorBlue: RGBColor = RGBColor.Blue;
console.log('colorBlue:', colorBlue);

enum Cities1 { Mumbai = 10, Delhi = 20, Kolkata = 30, Chennai = 40, Bangalore = 50
}
console.log('Cities:', Cities1);
console.log('Bangalore:',Cities1.Bangalore);

enum Cities2 { Mumbai = 'M', Delhi = 'D', Kolkata = 'K', Chennai = 'C', Bangalore
= 'B' }
console.log('Cities:', Cities2);
console.log('Bangalore:',Cities2.Bangalore);
```

4.8. Type Inference

- Defining variable type is optional, but sometimes Typescript infer or decide the type of variable while initialization only
- let details; // no type infer or decided/finalized as no value provided
- let details = 'Human'; // string (hover on variable-details to know/get data type)
- Once the data type is inferred or finalized by Typescript we cant assign other type values to it (will get an error `not assignable`)
- Types are inferred by TypeScript compiler when:
    - Variables are initialized
    - Default values are set for parameters
    - Function return types are determined

> **Syntax & Example**: Typescript

```
export {};

let details; //data type not infered
console.log('type of details:', typeof(details));

let cityName = 'Human'; // hover on cityName to check type details let cityName:
string
console.log('type of cityName:', typeof(cityName));

//Error - Type '90' is not assignable to type 'string'.ts(2322), cant assign
number to string data type
//cityName = 90
```

4.9. Type Assertion (Type Casting)

- Type assertion allows Programmer to set the type of a value and tell the compiler not to infer
- Type assertion can be used when Programmer have a better understanding of the type of a variable (we may know a more accurate type) than what TypeScript can infer
- Type Assertion is similar to type casting in other languages
- `as` keyword is used for Type Assertion, or before value specify the type

> **Syntax & Example**: Typescript

```
export {};

let id = 100;
let employeeID = &lt;number&gt; id;

let name = 'Dinanath';
let fName = 'Dinanath';
console.log('fName data type:',typeof(fName));

fName = fName as string;
console.log('fName data type:',typeof(fName));
```

4.10. Functions

- Functions are the primary fundamental building block in JavaScript
- A function is a set of statements to perform a specific task
- Functions make the code more readable, maintainable, and reusable
- Typescript functions help to specify the data type of function parameters as well as a function return type
- In JavaScript every parameter is assumed to be required if no parameter passed - throws an error
- With Typescript, we can denote/mark/treat parameter optional by adding `? question mark` symbol
- ES6 allows function parameters to have/specify simple and intuitive default values

- It simply means that if no arguments are provided to function call these default parameters values will be used

Syntax & Example: Typescript

```
export {};

// specify data types of parameters
function sum(num1: number, num2: number) {
  console.log(num1 + num2);
}

sum(10, 20);

//error - only numbers allowed
// sum('one','two');

// specify data types of parameters as well function return type
function showGreetings(message: string): string {
  return message;
}
let message1 = showGreetings('Welcome to Typescript');
console.log(message1);
```

Syntax & Example: Typescript

```
export { };

// specify data types of parameters, use ? to denote optional parameters
function sum(num1: number, num2?: number) {
  if (num2) {
    console.log(num1 + num2);
  } else {
    console.log(num1);
  }
}

sum(10, 20);
sum(100);
```

Syntax & Example: Typescript

```
export { };

// specify data types of parameters, use ? to denote optional parameters, one can
also specify default values
function sum(num1: number, num2: number = 50) {
  console.log(num1 + num2);
```

```
}

sum(10, 20);
sum(100);
```

4.11. Interface

- Interfaces are the most flexible way to define the type of variables or properties of an object
- The `interface` keyword is used to declare an interface
- The interface is a structural/syntactical contract that defines the types, contract (rule) in your application
- It defines the types and syntax for classes to follow, Classes derived from an interface must follow the structure provided by their interface
- Any optional property defined in an interface are represented with `? question mark` sign (city?: string)

> **Syntax & Example**: `Typescript`

```
export {};

interface Person {
  firstName: string;
  lastName: string;
  age?: number
}

function showDetails (_person: Person) {
  console.log(`Person Details: ${_person.firstName} ${_person.lastName}
${_person.age}`);
}

let person1 = {
  firstName: 'Dinanath',
  lastName: 'Jayaswal',
}

showDetails(person1);

let person2 = {
  firstName: 'Dino',
  lastName: 'J.',
  age: 40
}

showDetails(person2);
```

4.12. Class constructor

- TypeScript is object-oriented JavaScript
- In OOPS (Object Oriented Programming Structure) a class is a blueprint for creating objects

- JavaScript has been primarily a functional programming language works mainly on prototype-based inheritance, Functions are used to build reusable components
- TypeScript introduced classes to avail the benefit of object-oriented techniques like interfaces, inheritance, encapsulation, and abstraction
- Typescript helps to write constructors with shorter syntax, prefix parameters with access modifiers
- Class-based inheritance needs `extends` keyword and main class properties and method can be accessed with `super` keyword

Syntax & Example: `Typescript`

```
export {};

class Greetings {
  personName: string;

  constructor(name: string) {
    this.personName = name;
  }

  showGreet() {
    console.log(`Welcome, have a great Day ${this.personName}!`)
  }

}

let person1 = new Greetings('Dinanath');
console.log('person1.personName:',person1.personName);
person1.showGreet();
```

Syntax & Example: `Typescript - class based inheritance`

```
// class-based inheritance
class Person extends Greetings {

  constructor (pName: string) {
    super(pName);
  }

  doTask() {
    console.log('Person doing some task');
  }

}

let person1 = new Person('Sagar');
console.log('person1.name:',person1.name);
person1.showGreet();
person1.doTask();
```

4.13. Access Modifiers

- Access modifiers are keywords that set the accessibility/availability/visibility of properties and methods
- There are three types of access modifiers in TypeScript:
    1. **public** -
        - A public data member has universal accessibility, Data members in a class are public by default
        - By default, all members of a class in TypeScript are public
        - All the public members can be accessed anywhere without any restrictions
    2. **private** -
        - The private access modifier ensures that class members are visible only to that class and are not accessible outside the containing class
        - In case An external class member tries to access a private member, the compiler throws an error
    3. **protected** -
        - The protected access modifier is similar to the private access modifier, except that protected members can be accessed using their deriving classes
        - A protected data member is accessible by the members within the same class as that of the former and also by the members of the child classes

> **Syntax & Example**: `Typescript`

```
export {};

class Greetings {
  public name: string;
  private privateName: string;
  protected protectedName: string;

  constructor(_name: string) {
    this.name = _name;
  }

  showGreet() {
    console.log(`Welcome, have a great Day ${this.name}!`)
  }

}

let greet1 = new Greetings('Dinanath');
console.log('greet1.name:',greet1.name);
greet1.showGreet();

// error - private property/variable not available outside
// console.log(greet1.privateName);

console.log(`// ------------------------------`);

// class-based inheritance
class Person extends Greetings {
```

```
  constructor (pName: string) {
    super(pName);
  }

  doTask() {
    console.log('Person doing some task');
    // error - private property/variable not available outside
    // console.log(greet1.privateName);

    // protected property/variable can be accessed in derived/child class
    console.log('this.protectedName:',this.protectedName);
  }

}

let person1 = new Person('Sagar');
console.log('person1.name:',person1.name);
person1.showGreet();
person1.doTask();
```

4.14. Decorator

- Decorators are experimental features in Typescript and may not be part of feature releases
- Decorators are a special type of declarations, metadata attached to classes, methods, and properties
- Decorators use the form `@expression`, where `expression must evaluate to a function` that will be called at runtime with information about the decorated declaration
- To enable experimental support for decorators, enable the `experimentalDecorators` compiler option either on the `command line` or in `tsconfig.json`:
- **command line**: `tsc --target ES5 --experimentalDecorators`
- **tsconfig.json:**

```
{
    "compilerOptions": {
        "target": "ES5",
        "experimentalDecorators": true
    }
}
```

- Decorators begins with `@ (at the rate sign)`

> **Syntax & Example**: `Typescript`

```
@Component({
  selector: 'user',
  templateUrl: 'user.component.html',
  styleUrls: ['user.component.scss']
})
```

Section 5. Reference

5.1. Websites / Blogs

- Official website: http://www.typescriptlang.org/
- Source code: https://github.com/Microsoft/TypeScript

5.2. Books

- https://livebook.manning.com/#!/book/typescript-quickly/chapter-1/v-5/1

Section 6. What's Next Step?

Many Congratulations! You have done it. Thank you for joining me for `Typescript fundamentals for beginners`. I hope you will start checking/testing and incorporating these features into your code right away. I hope now you have a solid understanding of Typescript new features and usability. Your next step could be looking into Typescript documentation. Now you are fully up to date and set to start with any latest javascript framework/library like Angular, React, Node, Vue.